Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 7 of 7 returned.

☐ 1. Document ID: US 6104751 A

L19: Entry 1 of 7 File: USPT Aug 15, 2000

US-PAT-NO: 6104751
DOCUMENT-IDENTIFIER: US 6104751 A

TITLE: Apparatus and method for decompressing high definition pictures

DATE-ISSUED: August 15, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Artieri; Alain	Meylan			FR

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
SGS-Thomson Microelectronics S.A.	Gentilly			FR	03

APPL-NO: 08/ 329945 [PALM]
DATE FILED: October 26, 1994

PARENT-CASE:
CROSS REFERENCE TO RELATED APPLICATIONS This application is a continuation of application Ser. No. 08/247,996, filed May 24, 1994, entitled Picture Processing System now U.S. Pat. No. 5,579,052.

FOREIGN-APPL-PRIORITY-DATA:

COUNTRY	APPL-NO	APPL-DATE
FR	93 13294	October 29, 1993

INT-CL: [07] H04 N 7/12

US-CL-ISSUED: 375/240; 348/384, 348/721, 348/725, 348/701, 348/390
US-CL-CURRENT: 375/240.14; 348/701, 348/721, 348/725

FIELD-OF-SEARCH: 348/390, 348/402, 348/403, 348/384, 348/421, 348/721, 348/408, 348/725, 348/699, 348/701, 348/416

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

h e b b g e e e f e e ef b e

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4352157	September 1982	Namimoto et al.	364/200
4462074	July 1984	Linde	364/200
4800441	January 1989	Sato	358/261
5138447	August 1992	Shen et al.	348/390
5212742	May 1993	Normile et al.	382/56
5216503	June 1993	Paik et al.	348/390
5357282	October 1994	Lee	348/403
5379070	January 1995	Retter et al.	348/403

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
0221741	May 1987	EP	
0554586	August 1993	EP	
3045106	June 1987	DE	
2187577	September 1987	GB	
WO 9111074	July 1991	WO	

OTHER PUBLICATIONS

Real-Time Parallel and Fully Pipelined Two-Dimensional DCT Lattice Structures with Application to HDTV Systems, Chiu et al, IEEE Trans. on Circuits and Systems for Video Technology, vol. 2, No. 1, Mar. 1992, pp. 25-37.
"A MPEQ Decoder ASIC for Compact Disc Interactive", Cor Schepens, IEEE Publications, .COPYRGT.1992, pp. 301-304.
Proceedings of the 6th Mediterranean Electrotechnical Conference May 1991, Ljubljana, Slovenia pp. 428-431, XP289486, E.J. Laloya-Monzon et al. "DSP Parallel Architecture For Image Compression".
Digital Image Processing Applications, Los Angeles, CA, Jan. 17-20, 1989, 140-147, Yusheng. T. Tsai, "Real-time architecture for error-tolerant color picture compression."
IEEE Colloquim on Parallel Architectures for Image Processing Applications, Digest No. 086, London, UK, Apr. 22, 1991, M.N. for Adaptive Transform Coding Algorithms.

ART-UNIT: 273

PRIMARY-EXAMINER: Le; Vu

ATTY-AGENT-FIRM: Wolf, Greenfield & Sacks, P.C. Morris; James H. Galanthay; Theodore E.

ABSTRACT:

A system for processing compressed data corresponding to pictures includes a decoding mechanism, providing a picture memory with decoded picture data. The decoding mechanism requires, for decoding a current block of a previously decoded picture. A plurality of decoders are associated with respective picture memories, each storing a specific slice of corresponding blocks of a plurality of pictures, as well as at least one margin which is liable to be a predictor block serving to decode a block of the specific slice.

h e b b g e e e f e e ef b e

32 Claims, 10 Drawing figures

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw De

---

6 2. Document ID: US 5805878 A

L19: Entry 2 of 7 File: USPT Sep 8, 1998

US-PAT-NO: 5805878
DOCUMENT-IDENTIFIER: US 5805878 A
** See image for Certificate of Correction **

TITLE: Method and apparatus for generating branch predictions for multiple branch instructions indexed by a single instruction pointer

DATE-ISSUED: September 8, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Rahman; Monis	San Jose	CA		
Yeh; Tse-Yu	Milpitas	CA		
Poplingher; Mircea	Campbell	CA		
Scafidi; Carl C.	Sunnyvale	CA		
Choubal; Ashish	Santa Clara	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Intel Corporation	Santa Clara	CA			02

APPL-NO: 08/ 792115 [PALM]
DATE FILED: January 31, 1997

INT-CL: [06] G06 F 9/40

US-CL-ISSUED: 395/586
US-CL-CURRENT: 712/239

FIELD-OF-SEARCH: 395/376, 395/586, 395/587, 395/580

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
5093778	March 1992	Favor et al.	395/375
5687360	November 1997	Chang	395/587

ART-UNIT: 274

PRIMARY-EXAMINER: Eng; David Y.

h e b b g e e e f e e ef b e

ATTY-AGENT-FIRM: Blakely, Sokoloff, Taylor & Zafman

ABSTRACT:

A method and apparatus for generating respective branch predictions for first and second branch instructions, both indexed by a first instruction pointer, is disclosed. The apparatus includes dynamic branch prediction circuitry for generating a branch prediction based on the outcome of previous branch resolution activity, as well as static branch prediction circuitry configured to generate a branch prediction based on static branch prediction information. Prediction output circuitry, coupled to the both the dynamic and static branch prediction circuitry, outputs the respective branch predictions for the first and second branch instructions in first and second clock cycles to an instruction buffer (or "rotator"). Specifically, the prediction output control circuitry outputs the branch prediction for the second branch instruction in the second clock cycle and in response to the initiation of a recycle stall during the first clock cycle. The receipt of the predictions and their corresponding branch instructions at the instruction buffer is thus synchronized, and the opportunity of generating a prediction for the second branch instruction is not lost.

33 Claims, 13 Drawing figures

---

6 3. Document ID: US 5781750 A

L19: Entry 3 of 7 File: USPT Jul 14, 1998

US-PAT-NO: 5781750
DOCUMENT-IDENTIFIER: US 5781750 A

TITLE: Dual-instruction-set architecture CPU with hidden software emulation mode

DATE-ISSUED: July 14, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Blomgren; James S.	San Jose	CA		
Richter; David E.	San Jose	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Exponential Technology, Inc.	San Jose	CA			02

APPL-NO: 08/ 179926 [PALM]
DATE FILED: January 11, 1994

INT-CL: [06] G06 F 9/455

US-CL-ISSUED: 395/385; 395/500, 395/570
US-CL-CURRENT: 712/209; 703/26, 712/229

FIELD-OF-SEARCH: 395/385, 395/500, 395/800.43, 395/570, 395/800.23, 395/800.41

h e b b g e e e f e e ef b e

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
3764988	Ocbo er 1973	Onishi	395/581
4377844	March 1983	Kaufman	711/220
4456954	June 1984	Bullions, III et al.	711/207
4514803	April 1985	Agnew et al.	395/500
4538241	August 1985	Levin et al.	711/207
4633417	December 1986	Wilburn et al.	364/550
4691278	September 1987	Iwata	395/500
4763242	August 1988	Lee et al.	395/500
4780819	October 1988	Kashiwagi	395/500
4794522	December 1988	Simpson	395/500
4812975	March 1989	Adachi et al.	395/500
4821187	April 1989	Ueda et al.	395/596
4841476	June 1989	Mitchell et al.	395/500
4942519	July 1990	Nakayama	395/290
4972317	November 1990	Buonomo et al.	395/568
4991081	February 1991	Boshart	711/3
4992934	February 1991	Portanova et al.	395/385
5077654	December 1991	Ohtsuki	711/203
5077657	December 1991	Cooper et al.	395/500
5097407	March 1992	Hino et al.	395/385
5136696	August 1992	Beckwith et al.	395/587
5167023	November 1992	de Nicolas et al.	395/527
5210832	May 1993	Maier et al.	395/568
5222223	June 1993	Webb, Jr. et al.	711/140
5226164	July 1993	Nadas et al.	395/385
5230045	July 1993	Sindhu	711/203
5230069	July 1993	Brelsford et al.	711/6
5255384	October 1993	Sachs et al.	711/207
5291586	March 1994	Jen et al.	395/500

OTHER PUBLICATIONS

"High Performance Dual Architecture Processor", IBM Technical Disclosure Bulletin, vol. 36, No. 2, Feb. 1993, pp. 231-234.
Tanenbaum, "Structured Computer Organization", Prentice-Hall 1984, pp. 10-12.
Combining both micro-code and Hardwired control in RISC by Bandyophyay and Zheng,, Sep. 1987 Computer Architecture News pp. 11-15.
Combining RISC and CISC in PC systems By Garth, Nov. 1991 IEEE publication (?) pp. 10/1 to 10/5.

ART-UNIT: 274

PRIMARY-EXAMINER: Lall; Parshotam S.

ASSISTANT-EXAMINER: Vu; Viet

h e b b g e e e f e e ef b e

ATTY-AGENT-FIRM: Auvinen; Stuart T.

ABSTRACT:

A dual-instruction-set CPU is able to execute x86 CISC (complex instruction set computer) code or PowerPC RISC (reduced instruction set computer) code. Three modes of operation are provided: CISC mode, RISC mode, both called user modes, and emulation mode. Emulation mode is entered upon reset, and performs various system checks and memory allocation. A special emulation driver is loaded into a portion of main memory set aside at reset. Software routines to emulate the more complex instructions of the CISC architecture using RISC instructions are also loaded into the emulation memory. A TLB is enabled, and translation tables and drivers are set up in the emulation memory. All TLB misses, even in the user modes, will cause entry to a translator drive in emulation mode. Since the TLB is always enabled for the user modes, and all misses are handled by the emulation code, the emulation code can set aside a portion of memory for itself and insure that the user programs never have access to the emulation memory. Thus the programs, including operating systems, in CISC or RISC mode are unaware of emulation memory or even the existence of emulation mode.

20 Claims, 3 Drawing figures

---

4. Document ID: US 5721855 A

L19: Entry 4 of 7 File: USPT Feb 24, 1998

US-PAT-NO: 5721855
DOCUMENT-IDENTIFIER: US 5721855 A

TITLE: Method for pipeline processing of instructions by controlling access to a reorder buffer using a register file outside the reorder buffer

DATE-ISSUED: February 24, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Hinton; Glenn J.	Portland	OR		
Papworth; David B.	Beaverton	OR		
Glew; Andrew F.	Hillsboro	OR		
Fetterman; Michael A.	Hillsboro	OR		
Colwell; Robert P.	Portland	OR		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Intel Corporation	Santa Clara	CA			02

APPL-NO: 08/ 679182 [PALM]
DATE FILED: July 12, 1996

PARENT-CASE:

h e b b g e e e f e e ef b e

This is continuation of application Ser. No. 08/205,021, filed Mar. 1, 1994, now abandoned.

INT-CL: [06] G06 F 9/38

US-CL-ISSUED: 395/394; 395/393
US-CL-CURRENT: 712/218; 712/217

FIELD-OF-SEARCH: 395/394, 395/393

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4722049	January 1988	Lahti	395/800.03
4773041	September 1988	Hassler et al.	395/421.03
4791557	December 1988	Angel et al.	395/591
5129067	July 1992	Johnson	395/389
5185871	February 1993	Frey et al.	395/381
5261071	November 1993	Lyon	395/467
5280615	January 1994	Church et al.	395/674
5323489	June 1994	Bird	395/494
5355457	October 1994	Shebanow et al.	395/394
5363495	November 1994	Fry et al.	395/555
5377341	December 1994	Kaneko et al.	395/496
5463745	October 1995	Vidwans et al.	395/394
5471633	November 1995	Colwell et al.	395/800.23
5548776	August 1996	Colwell et al.	395/393
5553256	September 1996	Fetterman et al.	395/393
5651125	July 1997	Witt et al.	395/394

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
2251320	January 1992	GB	

OTHER PUBLICATIONS

Mike Johnson; Superscalar Microprocessor Design; pp. 127-133, 139-142, 146, 199-200; 1991.
Popescu, Val; Schultz, Merle; Spracklen, John; Gibson, Gary; Lightner, Bruce; Isaman, David, "The Metaflow Architecture", IEEE Micro, Jun. 1991, p. Nos. 10-13 and 63-73.
Johnson, Mike, "Superscalar Microprocessor Design", Englewood Cliffs, N.J.,:Prentice-Hall, 1991 (only table of contents).

ART-UNIT: 235

PRIMARY-EXAMINER: Lall; Parshotam S.

h e b b g e e e f e e ef b e

ASSISTANT-EXAMINER: Coulter; Kenneth R.

ATTY-AGENT-FIRM: Blakely, Sokoloff, Taylor & Zafman

ABSTRACT:

A pipelined method for executing instructions in a computer system. The present invention includes providing multiple instructions as a continuous stream of operations. This stream of operations is provided in program order. In one embodiment, the stream of operations is provided by performing an instruction cache memory lookup to fetch the multiple instructions, performing instruction length decoding on the instructions, rotating the instructions, and decoding the instructions. The present invention also performs register renaming, allocates resources and sends a portion of each of the operations to a buffering mechanism (e.g., a reservation station). The instruction cache memory lookup, instruction length decoding, rotation and decoding of the instructions, as well as the register renaming, are performed in consecutive pipestages.

The present invention provides for executing the instructions in an out-of-order pipeline. The execution produces results. In one embodiment, the instructions are executed by determining the data readiness of each of the operations and scheduling data ready operations. These scheduled data ready operations are dispatched to an execution unit and executed. The results are written back for use by other operations or as data output or indication. The determination of execution readiness, the dispatching and the execution, and writeback, are performed in consecutive pipestages.

The present invention also provides for retiring each of the continuous stream of operations in such a manner as to commit their results to architectural state and to reestablish sequential program order.

23 Claims, 53 Drawing figures

---

5. Document ID: US 5600837 A

L19: Entry 5 of 7 File: USPT Feb 4, 1997

US-PAT-NO: 5600837
DOCUMENT-IDENTIFIER: US 5600837 A

TITLE: Multitask processor architecture having a plurality of instruction pointers

DATE-ISSUED: February 4, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Artieri; Alain	Meylan			FR

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
SGS-Thomson Microelectronics S.A.	Saint Genis Pouilly			FR	03

h e b b g e e e f e e ef b e

APPL-NO: 08/ 248472 [PALM]
DATE FILED: May 24, 1994

FOREIGN-APPL-PRIORITY-DATA:

COUNTRY	APPL-NO	APPL-DATE
FR	93 06612	May 27, 1993

INT-CL: [06] G06 F 9/06

US-CL-ISSUED: 395/673; 364/228.2, 364/230.3, 364/DIG.1, 395/677
US-CL-CURRENT: 718/103; 718/107

FIELD-OF-SEARCH: 395/650, 395/700, 395/375, 364/DIG.1, 364/DIG.2

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4352157	September 1982	Namimoto et al.	
4462074	July 1984	Linde	
5016162	May 1991	Epstein et al.	364/200
5255384	October 1993	Sachs et al.	395/425

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
0221741	May 1987	EP	
2187577	September 1987	GB	

OTHER PUBLICATIONS

The 19th Annual International Symposium on Computer Architecture, May 19-21, 1992, Gold Coast Australia, "An Elementary Processor Architecture with Simultaneous Instruction Issuing From Multiple Threads", pp. 136- 45.
The 17th Annual International Symposium on Computer Architecture, May 28-31, 1990, Seattle, WA IEEE Computer Society Press, Los Alamitos, CA, "April: A Processor Architecture for Multiprocessing" Anant Agarwal, et al., pp. 104-114.
The 15th Annual International Symposium on Computer Architecture, May 30-Jun. 2, 1988, Honolulu, Hawaii, IEEE Computer Society, "MASA: A Multithreaded Processor Architecture for Parallel Symbolic Computing", Robert H. Halsted, Jr., et al., pp. 443-451.
1994 International Conference on Parallel and Distributed Systems, IEEE Computer Society Press, Los Alamitos, CA, "A Fast Switching Double Processing Architecture for Multi-Tasking Real-Time Systems", Tein-Hsiang Lin, et al., pp. 82-87.

ART-UNIT: 236

PRIMARY-EXAMINER: Kriess; Kevin A.

ASSISTANT-EXAMINER: Banankhah; Majid A.

h e b b g e e e f e e ef b e

ATTY-AGENT-FIRM: Driscoll; David M. Morris; James H. Dorney; Brett N.

ABSTRACT:

A processor architecture for executing a current task among a plurality of possible tasks. The architecture includes: a plurality of instruction pointers respectively associated with the tasks and each storing the address of the current instruction to be executed of the associated task, only one of these pointers being enabled at a time to supply an address to the memory; a priority level decoder including circuitry for assigning a predetermined priority level to each request signal and for enabling the instruction pointer associated with the active request signal having the highest priority level; and a mechanism for incrementing the content of the enabled instruction pointer and for reinitializing it at the start address of the associated program when its content reaches the end address of the associated program.

17 Claims, 5 Drawing figures

---

6. Document ID: US 5579052 A

L19: Entry 6 of 7 File: USPT Nov 26, 1996

US-PAT-NO: 5579052
DOCUMENT-IDENTIFIER: US 5579052 A

TITLE: Picture processing system

DATE-ISSUED: November 26, 1996

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Artieri; Alain	Meylan			FR

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
SGS-Thomson Microelectronics S.A.	Saint Genis Pouilly			FR	03

APPL-NO: 08/ 247996 [PALM]
DATE FILED: May 24, 1994

FOREIGN-APPL-PRIORITY-DATA:

COUNTRY	APPL-NO	APPL-DATE
FR	93 06612	May 27, 1993
FR	93 13293	October 29, 1993

INT-CL: [06] H04 N 7/30, H04 N 7/32

US-CL-ISSUED: 348/416
US-CL-CURRENT: 375/240.15

h e b b g e e e f e e ef b e

FIELD-OF-SEARCH: 348/416, 348/699, 382/56, 382/43, 375/245, 375/246, 375/253

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4800441	January 1989	Fumitaka Sato	358/261.1
5253078	October 1993	Balkanski et al.	348/416
5379356	January 1995	Purcell et al.	382/56

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
3545106	June 1987	DE	

OTHER PUBLICATIONS

Digital Image Processing Applications, Los Angeles, CA, Jan. 17-20, 1989, 140-147, Yusheng. T. Tsai, "Real-time architecture for error-tolerant color picture compression."
IEEE Colloquim on Parallel Architectures for Image Processing Applications, Digest No. 086, London, UK, Apr. 22, 1991, M. N. Chong, et al., "Pipeline Functional Algorithms, Data Partitioning for Adaptive Transform Coding Algorithms.""
A One Chip VLSI for Real Time Two-Dimensional Discrete Losine", Circuits & Systems, 1988 IEEE Internal Sypos, Artieri et al, pp. 701-704.
"A Realtime Image Processing Chip Set", Solid State Circuits, 1989 36th Conference, IEEE.
"Designing a High-Throughput VLC Decoder Parts I-II-Parallel Decoding Methods", Lin et al, IEEE Trans. in Circuits & Systems for Video technology, vol. 2, No. 2, Jun. 1992, pp. 187-206.

ART-UNIT: 265

PRIMARY-EXAMINER: Chin; Tommy P.

ASSISTANT-EXAMINER: Le; Vu

ATTY-AGENT-FIRM: Driscoll; David M. Morris; James H. Dorny; Brett N.

ABSTRACT:

A system that processes compressed data arriving in packets corresponding to picture blocks, the packets being separated by headers containing decoding parameters of the packets. A memory bus is controlled by a memory controller to exchange data between the processing elements and a picture memory. A pipeline circuit contains a plurality of processing elements. A parameter bus provides packets to be processed to the pipeline circuit, as well as the decoding parameters to elements of the system. The parameter bus is controlled by a variable length decoder that receives the compressed data from the memory bus and that extracts the packets and the decoding parameters therefrom.

h e b b g e e e f e e ef b e

13 Claims, 10 Drawing figures

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw De

---

6 7. Document ID: US 4325120 A

L19: Entry 7 of 7 File: USPT Apr 13, 1982

US-PAT-NO: 4325120
DOCUMENT-IDENTIFIER: US 4325120 A
** See image for Certificate of Correction **

TITLE: Data processing system

DATE-ISSUED: April 13, 1982

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Colley; Stephen R.	Aloha	OR		
Cox; George W.	Portland	OR		
Rattner; Justin R.	Aloha	OR		
Swanson; Roger C.	Portland	OR		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Intel Corporation	Santa Clara	CA			02

APPL-NO: 05/ 971661 [PALM]
DATE FILED: December 21, 1978

INT-CL: [03] G06F 13/00

US-CL-ISSUED: 364/200
US-CL-CURRENT: 711/202; 711/221

FIELD-OF-SEARCH: 364/2MSFile, 364/9MSFile

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
3648253	March 1972	Mullery et al.	364/200
3787818	January 1974	Arnold et al.	364/200
3905023	September 1975	Perpiglia	364/200
4044334	August 1977	Bachman et al.	364/200
4047161	September 1977	Davis et al.	364/200
4060849		Brenvenu et al.	364/200
4071890	January 1978	Pandeya	364/200
4084224	April 1978	Appell et al.	364/200

h e b b g e e e f e e ef b e

4104718 August 1978 Poublan et al. 364/200

ART-UNIT: 237

PRIMARY-EXAMINER: Springborn; Harvey E.

ATTY-AGENT-FIRM: Lamb; Owen L.

ABSTRACT:

A data processor architecture wherein the processors recognize two basic types of objects, an object being a representation of related information maintained in a contiguously-addresed set of memory locations. The first type of object contains ordinary data, such as characters, integers, reals, etc. The second type of object contains a list of access descriptors. Each access descriptor provides information for locating and defining the extent of access to an object associated with that access descriptor. The processors recognize complex objects that are combinations of objects of the basic types. One such complex object (a context) defines an environment for execution of objects accessible to a given instance of a procedural operation. The dispatching of tasks to the processors is accomplished by hardware-controlled queuing mechanisms (dispatching-port objects) which allow multiple sets of processors to serve multiple, but independent sets of tasks. Communication between asynchronous tasks or processes is accomplished by related hardware-controlled queuing mechanisms (buffered-port objects) which allow messages to move between internal processes or input/output processes without the need for interrupts. A mechanism is provided which allows the processors to communicate with each other. This mechanism is used to reawaken an idle processor to alert the processor to the fact that a ready-to-run process at a dispatching port needs execution.

56 Claims, 20 Drawing figures

Term	Documents
(15 AND 18).USPT.	7
(L15 AND L18).USPT.	7

h e b b g e e e f e e ef b e

L19: Entry 2 of 7 File: USPT Sep 8, 1998

DOCUMENT-IDENTIFIER: US 5805878 A
** See image for Certificate of Correction **
TITLE: Method and apparatus for generating branch predictions for multiple branch instructions indexed by a single instruction pointer

Abstract Text (1):
A method and apparatus for generating respective branch predictions for first and second branch instructions, both indexed by a first instruction pointer, is disclosed. The apparatus includes dynamic branch prediction circuitry for generating a branch prediction based on the outcome of previous branch resolution activity, as well as static branch prediction circuitry configured to generate a branch prediction based on static branch prediction information. Prediction output circuitry, coupled to the both the dynamic and static branch prediction circuitry, outputs the respective branch predictions for the first and second branch instructions in first and second clock cycles to an instruction buffer (or "rotator"). Specifically, the prediction output control circuitry outputs the branch prediction for the second branch instruction in the second clock cycle and in response to the initiation of a recycle stall during the first clock cycle. The receipt of the predictions and their corresponding branch instructions at the instruction buffer is thus synchronized, and the opportunity of generating a prediction for the second branch instruction is not lost.

Brief Summary Text (2):
The present invention pertains to the generation of branch predictions in a microprocessor. More particularly, the present invention relates to a method and apparatus for generating branch predictions for multiple branch instructions indexed as a group by a single instruction pointer.

Brief Summary Text (9):
In order to expedite the branch prediction process, a branch prediction table (BPT) may be maintained within branch prediction circuitry to provide a cache for recent branch predictions. Referring now to FIG. 1, there is illustrated a portion of a microprocessor 10 incorporating branch prediction circuitry 12. The branch prediction circuitry 12 includes a branch prediction table (BPT) 14. The microprocessor 10 further includes an instruction pointer (IP) generator 16 and a fetcher 18. The fetcher 18 retrieves instructions, indexed by an instruction pointer (IP) generated by the IP generator 16, from an instruction cache 20 or from main memory via a bus interface unit 22. The fetcher 18 then propagates retrieved instructions to an instruction buffer (IB) 24, where the instructions are buffered prior to being forwarded to the instruction decode and execute circuitry 26.

Brief Summary Text (10):
As illustrated in FIG. 1, an instruction pointer, identifying a new instruction to be retrieved, is presented simultaneously to the fetcher 18 and to the BPT 14. While the fetcher 18 retrieves the instruction indexed by the instruction pointer, a determination is made as to whether an entry for the instruction referenced by the instruction pointer is present in the BPT 14 (i.e. whether there is a "hit" in the BPT 14). This determination may be made using a tag value, which comprises part of the instruction pointer. Initially, the BPT 14 is empty, and a MISS signal is generated. An instruction pointer (IP) increment unit 28 receives the MISS signal,

and increments the instruction pointer to identify the next sequential instruction of the program in memory. The incremented instruction pointer is then propagated to the instruction pointer generator 16, which in turn outputs the incremented instruction pointer. In other words, when the BPT 14 generates a MISS signal, the microprocessor 10 assumes that either the instruction indexed by the instruction pointer is not a branch instruction or, if it is a branch instruction, that the branch instruction will not be taken. The incremented instruction pointer is then propagated from the instruction pointer generator 16 back to the fetcher 18 and the BPT 14, and the process described above is repeated. Instructions retrieved by the fetcher 18 are forwarded to the instruction buffer 24 (also termed a "rotator"), and then propagated to the instruction decode and execute circuitry 26. The instruction decode and execute circuitry 26 will encounter and process a number of branch instructions in the course of executing a program. The instruction decode and execute circuitry 26 includes branch resolution circuitry 27, which is responsible for the resolution of qualifying conditions presented by branch instructions. For each branch instruction encountered, the branch resolution circuitry 27 causes the BPT allocation circuitry to update the BPT 14 contents by allocating an entry within the BPT 14 to the relevant branch instruction. The BPT allocation circuitry 30 furthermore writes prediction information into each entry in the BPT 14, which can then be utilized by a branch prediction logic unit 32 to predict whether the relevant branch instruction will be "taken" or "not taken" when next encountered in the instruction stream. The branch prediction logic unit 32 may indicate a prediction within an entry in the BPT 14, for example, by setting a prediction bit to one or zero.

Brief Summary Text (11):
As execution of a computer program continues, it will be appreciated that more and more entries are allocated within the BPT 14. Eventually, an instruction pointer (or at least a tag portion thereof) from the IP generator 16 will correspond to an entry within the BPT 14, and a hit will have occurred. The BPT 14 then outputs a prediction of either "taken" or "not taken" based on the state of the prediction bit discussed above. If the prediction is "not taken", then the IP increment unit 28 is activated to increment the IP by one, and to propagate a speculative instruction pointer to the IP generator 16. If the prediction is "taken", then a branch target address prediction unit 34 is activated to provide a speculative branch target address. The branch target address is specified within the branch instruction itself. The branch target address generated by the prediction unit 34 is propagated to the IP generator 16, from which it is re-circulated to the fetcher 18 and the BPT 14.

Brief Summary Text (13):
The BPT 14 provides a cache for prediction information for branch instructions of an executing computer program. It will be appreciated that the BPT 14 is of limited size, and that entries may accordingly need to be de-allocated in order to make room for new entries corresponding to newly encountered branch instructions. It may thus occur that the BPT 14 receives an instruction pointer corresponding to a previously predicted branch instruction for which there is no entry in the BPT 14, as a result of the appropriate entry having been de-allocated. In this case, the branch prediction circuitry 12 may nonetheless generate a prediction utilizing other means associated with the branch prediction circuitry 12. In other words, the branch prediction process is not limited to branch instructions having entries allocated within the BPT 14.

Brief Summary Text (14):
With the appearance of instruction caches which store two or more instructions per cache line, the above described structures encounter a number of inadequacies. In architectures including instruction cache lines each capable of storing more than a single instruction, the fetcher is capable of simultaneously retrieving all instructions contained in an instruction cache line, thus providing a performance advantage. To achieve this, the instruction pointer is interpreted as indexing a

cache line storing multiple instructions. In such architectures, instructions may further be included in data structures referred to in the art as "bundles" and a single cache line may include two or more bundles, each bundle including one or more instructions. The fetcher 18 then issues instructions to the instruction buffer 24. It will be appreciated that, where the instruction pointer is used simultaneously to index multiple instructions, the instruction pointer must be advanced by more than a single increment to reference a subsequent group of multiple instructions (which may be stored in a single cache line). Accordingly, it is conceivable that the instruction pointer, when advanced to point to a next instruction cache line, may skip, or jump, an entry in the BPT 14. For example, consider the scenario in which consecutive instruction cache lines are respectively index by the values IP and IP+2. In this case, an entry in the BPT 14 corresponding to the value IP+1 may be overlooked as the instruction pointer is advanced from the value IP to the value IP+2. This is undesirable, as the performance benefit that may have resulted from a prediction for the branch instruction at IP+1 is lost.

Brief Summary Text (17):
According to a first aspect of the invention there is provided a computer-implemented method of generating respective branch predictions, in a microprocessor, for first and second branch instructions. Both the first and second branch instructions are indexed by a first instruction pointer. The method requires generating a first branch prediction for the first branch instruction. If the first branch is predicted as being not taken, then a number of actions are performed. In a first clock cycle, a recycle stall is initiated during which the first instruction pointer is stalled. The first branch prediction is then propagated to an instruction buffer. A second branch prediction for the second branch instruction is then generated and, in a second clock cycle, the second branch prediction is propagated to the instruction buffer in response to the initiation of the recycle stall in the first clock cycle.

Brief Summary Text (24):
The microprocessor may also include an instruction cache having an instruction cache line, indexed by the first instruction pointer, storing both the first and second branch instructions. The recycle stall may furthermore be initiated if the both the first and second branch instructions are identified as being branch instructions. The microprocessor may also include a branch prediction table having an entry for the first branch instruction, the branch prediction table entry including a double branch indicator identifying both the first and second branch instructions as branch instructions. The first and second branch instructions may also each be identified as being a branch instruction by static branch prediction circuitry.

Brief Summary Text (26):
According to a second aspect of the invention there is provided apparatus for generating respective branch predictions for first and second branch instructions, both indexed by a first instruction pointer. The apparatus includes dynamic branch prediction circuitry configured to generate a branch predict on based on the outcome of previous branch resolution activity, as well as static branch prediction circuitry configured to generate a branch prediction based on static branch prediction information. Prediction output circuitry, coupled to the both the dynamic and static branch prediction circuitry, outputs the respective branch predictions for the first and second branch instructions in first and second clock cycles to an instruction buffer (or "rotator"). Specifically, the prediction output control circuitry outputs the branch prediction for the first branch instruction in the first clock cycle, and outputs the branch prediction for the second branch instruction in the second clock cycle and in response to the initiation of the recycle stall in the first clock cycle.

Brief Summary Text (28):
The prediction output control circuitry may include recycle stall circuitry which

initiates a recycle stall, during which the first instruction pointer is stalled, when the instruction pointer indexes the first and second branch instructions and when the first branch instruction is predicted as being not taken.

Brief Summary Text (29):
According to a third aspect of the invention there is provided apparatus for performing branch predictions for first and second branch instructions, both indexed by a first instruction pointer. The apparatus includes branch prediction circuitry which generates first and second branch predictions for the first and second branch instructions respectively, and prediction output control circuitry configured, if the first branch instruction is predicted as being not taken, to perform two primary functions. Specifically, the output control circuitry propagates the first branch prediction to an instruction buffer in a first clock cycle (for example in response to the initiation of a recycle stall during which the first instruction pointer is stalled), and then propagates the second branch prediction to the instruction buffer in a second clock cycle (for example in response to the initiation of the recycle stall in the first clock cycle)

Drawing Description Text (8):
FIG. 6 is a diagrammatic representation of an instruction pointer generated by an instruction pointer generator.

Drawing Description Text (12):
FIGS. 10A-10D are flowcharts illustrating a method of generating branch predictions for multiple branch instructions indexed, as a group, by a single instruction pointer.

Detailed Description Text (2):
A method and apparatus for generating branch predictions for multiple branch instructions within a instruction cache line, indexed by a single instruction pointer, are described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be evident, however, to one skilled in the art that the present invention may be practiced without these specific details.

Detailed Description Text (5):
Referring now to FIG. 3, a more detailed overview of the processor 14 of FIG. 1 is shown in block diagram form. The processor 44 comprises a bus interface unit 62, which provides the interface between the processor 44 and the bus 32 of the computer system 40. The bus interface unit 62 is coupled to allow a fetcher 64 and a data cache 66 to access the main memory 46 of the computer system 40. Also coupled to the fetcher 64 is an instruction cache 68, also referred to in the art as a code cache. The fetcher 64 retrieves instructions indexed by an instruction pointer (which may for example be generated within a branch prediction unit 65) from the instruction cache 68 or from main memory 46 via the bus interface unit 62. The branch prediction unit 65 speculatively predicts target addresses for branch instructions retrieved by the fetcher 64, and outputs an instruction pointer dependent on the speculative branch target prediction process. The fetcher 64 propagates instructions retrieved from either the instruction cache 68, or the main memory 46, to an instruction buffer 70, from which a decoder 72 retrieves instructions, when required, for decoding. As the rate at which instructions are retrieved by the fetcher 64 is not necessarily equal to the rate at which instructions are consumed by the decoder 72, the instruction buffer 70 provides decoupling between the fetcher 64 and the decoder 72. To this end, it is convenient to regard the fetcher 64, the instruction cache 68 and the bus interface unit 62 as comprising a front end of the processor 44, which is decoupled from the back end of the processor 44, comprising the remaining circuitry described below, by the instruction buffer 70. A microcode read only memory (ROM) 74 is coupled to the decoder 72 and receives entry-points from the decoder 72. The decoder 72 and a

microcode ROM 74 supply microcode instructions to a control unit 76. The control unit 76 supplies an integer pipeline execution unit 78, and a floating-point pipeline execution unit 80, with integer and floating-point instructions, respectively, for execution. The integer pipeline execution unit 78 and the floating-point pipeline execution unit 80 have access to the data cache 66 when executing the instructions. The integer pipeline execution unit 78 and the floating-point pipeline execution unit 80 are furthermore both coupled to memory management logic, which comprises a segment translator 82 and a page translator 84, the page translator 84 incorporating a translation lookaside buffer (TLB) 86. To perform the requisite functions, the integer pipeline execution unit 78 and the floating-point pipeline execution unit 80 each contain sets of address generation logic, arithmetic logic and cache memory interfaces.

Detailed Description Text (7):
FIG. 4 shows microprocessor circuitry, indicated generally at 100, including branch prediction circuitry 102, a fetcher 104, an instruction buffer 106 (also known in the art as a "rotator"), instruction decode and execute circuitry 108, a bus interface unit 110 and an instruction cache 116. The instruction decode and executed circuitry 108 may include a number of the microprocessor functional units described with reference to FIG. 3. The microprocessor circuitry 100 shown in FIG. 4 has a number of broad functional similarities to the microprocessor circuitry 10 shown in FIG. 1, but differs both structurally and functionally in a number of important respects. Firstly, the microprocessor circuitry 100 utilizes an instruction pointer which indexes multiple branch instructions as a group. In one embodiment, each of the multiple instructions is included within a data structure known as a "bundle", each bundle furthermore being divisible into three sub-structures termed "syllables". For the purposes of this specification, reference will merely be made to the term "instruction", with the understanding that an instruction may be incorporated within a "bundle". Furthermore, in the exemplary embodiment of the present invention described below, an instruction pointer is described as indexing a pair of instructions. However, it will readily be appreciated that the teachings of this invention can equally be applied to an architecture in which a single instruction pointer indexes any number of instructions. Referring now specifically to FIG. 4, the instruction cache 112 includes a number of cache lines, each cache line being sized to store multiple instructions. For the purposes of this specification, each instruction cache line is viewed as storing two instructions, each of which may be a branch instruction. A diagrammatic representation of an instruction cache line is provided in FIG. 7. As each instruction pointer references multiple instructions, a performance advantage is obtained within the microprocessor in that the multiple instructions referenced by the instruction pointer can simultaneously be retrieved by the fetcher 104. The fetcher 104 then dispatches instructions to the instruction buffer 106. This arrangement, whereby a single instruction pointer indexes at least two instructions for simultaneous (parallel) retrieval and serial dispatch, creates the potential for a number of difficulties with respect to branch prediction. Specifically, when two branch instructions are present within the indexed group of instructions, and a first one of these instructions is predicted as being "not taken", it is desirable to be able to perform a branch prediction for a second one of the branch instructions. It is also then desirable to output each of the two branch instructions in a serial manner, and in consecutive clock cycles, from the fetcher 104, and to synchronize the receipt of a prediction for each of the branch instructions with the receipt of a corresponding branch instruction at the instruction buffer 106. Accordingly, the order in which the branch prediction circuitry 102 outputs the respective predictions to the instruction buffer 106 must correspond to the order in which the fetcher 104 outputs the respective branch instructions to the instruction buffer 106. Furthermore, the timing with which the branch prediction circuitry 102 outputs each of the predictions must be synchronized with the timing with which the fetcher 104 outputs the respective branch instructions to the instruction buffer 106. The present invention teaches a method by which, when required, predictions for multiple branch instructions

indexed as a group by a single instruction pointer, may be outputted in a correctly ordered and synchronized manner for merging with appropriate branch instructions in the instruction buffer 106.

Detailed Description Text (8):
It will be appreciated that the above described ordering and timing requirements do not arise when only one branch instruction is included within multiple branch instructions referenced by a signal instruction pointer. Further, the above problems also do not arise when, although two branch instructions are indexed by a signal instruction pointer, the branch instruction located first within the instruction stream is predicted as "taken", as this prediction will result in the instruction stream being diverted (or being esteered) and accordingly rendering the second branch instruction of no consequence.

Detailed Description Text (9):
It will also be appreciated that each group of multiple instructions indexed by a specific instruction pointer may be located within a single cache line within the instruction cache 116 or, alternatively, may be located in main memory and retrieved by the fetcher 104 via the bus interface unit 110.

Detailed Description Text (12):
The dynamic branch prediction circuitry 114 is so termed as it dynamically monitors and updates prediction information for branch instructions encountered in an instruction stream, based on the resolution of qualifying conditions presented by each branch instruction and on the accuracy of earlier prediction activities. A number of structures and methods for achieving dynamic bran h prediction exists, and the above described exemplary embodiment is merely one example of such a structure. Referring specifically to FIGS. 4 and 5, the dynamic branch prediction circuitry 114 includes a branch prediction table (BPT) 120, branch prediction table allocation circuitry 122 and a branch prediction logic unit 124. As described above, the BPT 120 provides a cache of predictions for branch instructions most recently encountered in an instruction stream. The BPT 120 may be implemented as a four-way, set associative memory, as more fully described in co-pending U.S. patent application Ser. No. 08/785,199, entitled "Method and Apparatus for Performing Reads of Related Data from a Set-Associative Cache Memory". The BPT 120 includes a number of entries for recently encountered branch instructions. An example of a B T entry 126 is shown in FIG. 8, the entry 126 being shown to incorporate, inter alia, a tag 128, a bundle position bit 130, a prediction bit 132, and a double branch (DB) bit 134. In an architecture in which an instruction pointer indexes two instructions (or a "bundle"), the tag 128 may correspond to bits 1-15, and the bundle position bit 130 may correspond to bit 0, of the instruction pointer value for a corresponding branch instruction. Turning again to FIG. 4, the branch prediction circuitry 102 further includes an instruction pointer generator 140, an instruction pointer increment unit 142, and a target address prediction unit 144. The instruction pointer generator 140 is coupled to received inputs from the instruction pointer increment unit 142 and the target address prediction unit 144, as well as an exception signal 146 from exception generating circuitry (not shown). Referring to FIG. 6, there is shown a representation of an instruction pointer 150 outputted from the instruction pointer generator 140. In the illustrated embodiment, instruction pointer bits 1-15 (IP [15:1]) are examined within the instruction cache 112, and a hit is generated if the instruction pointer bits 1-15 correspond to a tag within a cache line within the instruction cache 112. In a similar manner, the instruction pointer 150 is propagated to the branch prediction table 120 within the dynamic branch prediction circuitry 114, and a hit is generated, merely for example, if instruction pointer bits 0-15 correspond to the combination of the tag 128 and the bundle position bit 130 of an entry 126 within the BPT 120. On the occurrence of such a hit, the BPT 120 outputs the current state of the prediction bit 132 which indicates the respective branch as being "taken" or "not taken". The state of the prediction bit 132 is propagated to the prediction output control circuitry 118 which, within the appropriate clock cycle and in an

appropriate order, forwards this prediction to the instruction pointer increment unit 142, the branch target address prediction unit 144 and the instruction buffer 108. Should the relevant branch be predicted as being "not taken", the instruction pointer increment unit 142 will increment the instruction pointer by an appropriate value, such as 2 in the present example. Alternatively, should the relevant branch be predicted as "taken", target address prediction unit 144 generates a branch target address. The target address prediction unit 144 generates this branch target address utilizing either target address information incorporated within the relevant instruction, or by accessing a table (not shown) which provides a history of branch addresses previously targeted by the relevant instruction.

Detailed Description Text (14):
In the event that there is no hit within the BPT 120 for a specific instruction pointer, the BPT 120 outputs a MISS signal 162 directly to the instruction pointer increment unit 142, which then increments the relevant instruction pointer by a predetermined value, and forwards this incremented instruction pointer to the instruction pointer generator 140.

Detailed Description Text (15):
As described with reference to FIG. 1, the BPT allocation circuitry 122 is responsible for allocating entries 126 within the BPT 120, and the branch prediction logic unit 124 is responsible for setting the prediction bit 132 within each entry 126 to either zero (0) or one (1) to indicate the branch as being either "taken" or "not taken" in accordance with a prediction algorithm employed by the branch prediction logic unit 124. The instruction pointer generator 140 issues an instruction pointer that, in the absence of a "taken" branch instruction, is incremented to refer to the next instruction cache line and, as each instruction cache line stores two instructions, is advanced by 2 each such iteration. Accordingly, the fetcher 104 is not concerned with the value of the instruction pointer bit 0 (IP[0]), as the retrieval of the instructions by the fetcher 104 occurs at a resolution lower than the highest resolution provided by the instruction pointer 150. For example, to register a hit in the instruction cache 112, the only condition is that instruction pointer bits 1-15 (IP [15:1]) correspond to the tag of an instruction cache line. If so, then the entire contents of the instruction cache line, which includes instructions at two consecutive instruction pointer values (for which IP [0]=0 and 1 respectively), is retrived by the fetcher 104. As the fetcher 104 retrieves instructions at a lower resolution than the resolution with which instructions are referenced in the BPT 120, it is conceivable that, when advancing the instruction pointer by two as described above, a branch instruction may be retrieved for which there is an entry in the BPT 120, but for which no hit in the BPT 120 occurs. Consider the example of two consecutive branch instructions referenced by instruction pointer values IP and IP+1. Assuming that the instruction pointer is incremented by two to reference a subsequent instruction cache line, only instruction pointer values IP and IP+2 will be presented to the BPT 120, and consequently a BPT entry for the branch instruction reference by IP+1 may be missed. To prevent this situation from occurring, an entry 126 within the BPT 120 is provided with the double branch (DB) bit 134, which may be set to indicate that another instruction within the group of multiple instructions indexed by the instruction pointer is also a branch instruction. Thus, continuing the above example, for an instruction pointer value IP, a hit within the BPT 120 for this value may reveal that another instruction within the indexed cache line (i.e. the instruction at IP+1) is a branch instruction. Accordingly, the BPT 120 is furthermore able to output a double branch signal 164 to the prediction output control circuitry 118, as shown in FIG. 5. When the double bit (DB) of an entry within the BPT 120 is set, and an intially encountered branch instruction is predicted as "not taken", a recycle stall occurs in which the instruction pointer is not advanced, so as to allow the BPT 120 to be reaccessed to determined whether there are BPT entries for any further branch instructions within the cache line.

Detailed Description Text (19):

The static prediction circuitry 116 also examines a branch instruction retrieved by the fetcher 104 to determine whether the relevant branch instruction includes a double branch indicator indicating that a further instruction, within the group of multiple instructions is indexed by the instruction pointer, is a branch instruction. Should the relevant branch instruction include a double branch indicator, the static branch prediction circuitry 116 outputs a double branch signal 182 to the prediction output control circuitry 118.

Detailed Description Text (21):
Referring to FIGS. 4 and 9, the prediction output control circuitry 118 includes recycle stall circuitry 170 and prediction selection circuitry 172. The prediction selection circuitry 172 in turn comprises multiplexers (MUXs) 174, 176 and 178. Broadly, the recycle stall circuitry 170 is responsible for the timing of the output of a prediction from the prediction selection circuitry 172, while the prediction selection circuitry 172 is responsible for the selection of an appropriate prediction from among the various predictions that may be received from the dynamic and static branch prediction circuitries 114 and 116. The need for the recycle stall circuitry 170 (also termed "reaccess circuitry") arises out of the capability of each instruction cache line within the instruction cache 112 to store more than one "bundle" of instructions. As each instruction cache line is indexed as a unit by an instruction pointer for retrieval by the fetcher 104, it may occur that the relevant instruction cache line includes two or more branch instructions. From a branch prediction viewpoint, assuming a sequentially first branch instruction is predicted as being "not taken" or misses the BPT, it is desirable to determine whether there are entries in the BPT for any of the other branch instructions. For this reason, a "recycle" or "reaccess" stall may be performed so as to allow the BPT to be indexed for the other branch instructions. The recycle stall circuitry 170 thus facilitates this process, and is also responsible for the timing of the output of predictions for two or more branch instructions indexed by a single instruction pointer.

Detailed Description Text (22):
The recycle stall circuitry 170 is coupled to receive double branch signals 164 and 182 from the dynamic and static branch prediction circuitries 114 and 116 respectively. The recycle stall circuitry 170 is also coupled to receive predictions 180, identifying a relevant branch instruction as being either "taken" or "not taken", for branch instructions indicating the existence of a double branch, from both the dynamic and static prediction circuitries 114 and 116. In the event that a branch instruction for a first branch instruction, processed by either the prediction circuitries 114 or 116, indicates the existence of a double branch (i.e. the existence of a second branch instruction in the group of indexed instructions), and the first branch instruction is predicted as being "not taken", then the recycle stall circuitry 170 asserts a recycle stall signal 186, indicating that the instruction pointer is to be stalled, so as to allow for a prediction to be outputted for the second branch instruction. In other words, the recycle stall circuitry 170 only asserts the recycle stall signal 186 on the occurrence of at least two branch instructions within a group of multiple instructions indexed by a single instruction pointer, and when a first branch instruction of the group is "not taken". The recycle stall signal 186 is not asserted when only one branch instruction is present in the indexed group as there is no need to attempt to obtain a dynamic or static branch prediction for any further branch instructions, nor to order the output of multiple predictions in this case. The recycle stall signal 186 is also not asserted when the first instruction is predicted to be "taken", and this prediction will result in a redirection of the instruction stream, and any subsequent branch instructions within the indexed group are of no concern, and will not be speculatively processed.

Detailed Description Text (23):
The MUXs 174 and 176 are each operable by the dynamic prediction valid signal 160, received from the dynamic prediction circuitry 114, to selected one of two inputs

for output to the MUX 178. Again assuming a situation in which at least two branch instructions, namely first and second branch instructions, are encountered within an instruction group indexed by the instruction pointer, the MUX 174 is coupled to receive any predictions for the first branch generated by the either the dynamic or static branch prediction circuitries 114 or 116. Similarly, the MUX 176 is coupled to receive any predictions for the second branch instruction which are generated by either the dynamic or static branch prediction circuitries 114 or 116. As long as the dynamic prediction signal 160 is asserted, the MUXs 174 and 176 will output only predictions received from the dynamic branch prediction circuitry 114, whereas if the dynamic prediction valid signal 160 is not asserted, the MUXs 174 and 176 will output predictions received from the static branch prediction circuitry 116. The MUX 178 is coupled to receive the outputs of the MUXs 174 and 176 as inputs, and is operable by the recycle stall signal 186 to select either of its inputs as an output to the instruction buffer 106. If the recycle stall signal 186 is asserted, the MUX 178 outputs a prediction for the second branch instruction, whereas a prediction for the first branch instruction will be outputted if the recycle stall signal 186 is not asserted.

Detailed Description Text (25):
The functioning of the above described computer system, microprocessor and circuitries will now be described with reference to the flowcharts shown in FIGS. 10A-10E, which detail an exemplary method 200 of generating branch predictions for multiple branch instructions indexed by a single instruction pointer for the purposes of retrieval by the fetcher 104. Assume for the purpose of explanation that the instruction pointer generator 140 outputs an instruction pointer for which there is a hit with reference to the instruction cache line 121, as shown in FIG. 7, of instruction cache 112. As illustrated, instruction cache line 121 includes a tag which is compared to bits 1-15 (IP[15:1]) of the instruction pointer to determine whether a hit occurs with reference to the instruction cache line 121. The instruction cache line 121 further includes two instructions, namely first and second instructions 125 and 127, either or both of which may be branch instructions. Each of the instructions 125 and 127 may or may not have corresponding entries in the BPT 120.

Detailed Description Text (26):
As a first step in the exemplary method 200, a determination is made at step 202 as to whether bit 0 (IP[0]) of the instruction pointer output is 0. If not (i.e. IP[0] =1), this indicates that the instruction pointer is indexing the second instruction 127 within the instruction cache line 121. This situation could occur as a result of a jump from another branch instruction, and typically would not arise where the instruction pointer is being advanced sequentially through instructions of a computer program. It will also be appreciated that the instruction pointer could similarly index a grouping of three or more instructions stored within an instruction cache line 121 of the instruction cache 112 or in main memory 16. In the present example, if the instruction pointer is indexing the second instruction 127 within the instruction cache line 121, the contents of this entire instruction line will be retrieved by the fetcher 104, which then dispatches only the second instruction to the instruction decode and execute circuitry 108, ignoring all preceding instructions. For the exemplary embodiment, clearly there are no subsequent branch instructions, in the indexed instruction cache line 121, for which predictions may be lost. Accordingly, the method 200 then teaches advancing the instruction pointer by one (1) at step 204. Assuming that the second instruction 127 was not a branch instruction predicted as being "taken", the instruction pointer will then index the instruction cache line following the cache line 121 in the instruction cache 112, and any entry in the BPT 120 for the first instruction stored in that cache line.

Detailed Description Text (27):
For the purposes of explaining the full functioning of the above described embodiment of the invention, it is further necessary to assume that the first and

second instructions 125 and 127 are both branch instructions, and further that the first branch instruction 125 is predicted as being "not taken". As detailed above, should the first instruction be predicted as being "taken", no prediction need be generated for the second branch instruction, as the instruction pointer will then jump to a branch target address.

Detailed Description Text (38):
If IP[0]=0 at step 202, a determination is made at step 206, and in clock cycle 0, as to whether there is a hit in the BPT 120 (or any other dynamic prediction structures) for the first instruction 125. If not, the dynamic prediction valid signal 160 is not asserted, thus causing MUXs 174 and 176 of the prediction selection circuitry 172 to select an input from the static prediction circuitry 116 as an output to the MUX 178. The method 200 then proceeds to step 208, as illustrated at in FIG. 10B, where a determination is made, again in clock cycle 0, as to whether there is a hit in the BPT 120 (or any other dynamic prediction structures) for the second instruction 127. If not, the method 200 proceeds to step 210 at which a static prediction is generated for the first instruction 125 by the static prediction circuitry 116, in a clock cycle 1. At step 212, the recycle stall signal 170 is asserted by the recycle stall circuitry 17 during clock cycle 1, after receiving an indication from the static branch prediction circuitry 116, via double branch signal 182 and prediction signal 180, that a double branch is present and that the first branch instruction 125 was "not taken". In response to the assertion of the recycle stall signal 186, the MUX 178 outputs the static prediction for the first instruction 125 to the instruction buffer 106 in clock cycle 1, at step 214. If the prediction for the first instruction is "not taken", a static prediction is generated for the second instruction 127, in clock cycle 2, as shown at step 216 in FIG. 10C. This prediction is then propagated to MUX 176 of the prediction output control circuitry 118. Again, as no dynamic prediction occurred, the dynamic prediction valid signal 160 remain de-asserted, causing MUX 176 to output the static branch prediction to MUX 178. At step 218, the recycle stall signal 186 is de-asserted, thereby causing MUX 178 to output the static prediction for the second instruction 127 to the instruction buffer 106, during clock cycle 2, at step 218. At step 220, if the prediction for the second instruction 127 was that it was "taken", then the method is diverted to step 222, at which a branch target address is generated by the target address prediction unit 144. The target address prediction unit 144 may utilize branch target address information contained within the second instruction 127 to generate a branch target address prediction for this instruction. Alternatively, should it be determined at step 2220 that the prediction for the second instruction 127 is that of "not taken", the instruction pointer is simply incremented by two at step 224.

Detailed Description Text (49):
In this scenario, a determination is made at step 206 that a hit occurs within the BPT 120 (or any other dynamic prediction structure) for the first instruction 125, and the method then proceeds through steps 226-238, as described above. At step 238, it is determined that there is a hit for the second branch instruction 127 in the BPT 120 (or any other dynamic prediction structure), and the method 200 accordingly proceeds to step 260, where a dynamic prediction for the second instruction 127 is then generated as shown in FIG. 10D. At step 262, and during clock cycle 2, the recycle stall signal is deasserted. The prediction for the second instruction 127 outputted from the MUX 178 at step 264 as a result of a recycle stall having occurred in the preceding clock cycle. In one exemplary embodiment, the occurrence of a recycle stall in the preceding clock cycle may be detected by a transition of the recycle stall signal (i.e. by the deassertion of the recycle stall signal). However, any other means by which a recycle stall in a preceding clock cycle may be detected can also be employed to trigger the output of the prediction for the second instruction 127. At step 266, a determination is made as to whether the second instruction 127 was predicted as being "taken" or "not taken". If instruction 127 was predicted as being "taken", a branch target address is generated at step 268 by the target address prediction unit 144, by accessing

history records maintain by the unit 144 or utilizing branch target information included within the second instruction 127 itself. Alternatively, should instruction 127 be predicted as being "not taken", the instruction pointer is incremented by 2 by the instruction pointer increment unit 142 at step 270.

Detailed Description Text (50):
In conclusion, the above described invention is advantageous in that it facilitates the output of predictions for multiple branch instructions, indexed as a group by an instruction pointer, in a serial manner and in the order in which the relevant branch instructions appear in the instruction stream. This is desirable as it allows branch instructions, which are issued serially from the fetcher 104, and corresponding predictions, to be received together at the instruction buffer 106. The serialization of the output of predictions is furthermore desirable in that the bus required for the propagation of prediction information, for example from the branch prediction circuitry 102 to the instruction buffer 106, need only be wide enough to accommodate prediction information for a single branch instruction. Accordingly, die space dedicated to such a bus is minimized.

Detailed Description Text (51):
Thus, a method and apparatus for generating branch predictions for multiple branch instructions, indexed by a single instruction pointer, have been described. Although the present invention has been described with reference to specific exemplary embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader spirit and scope of the invention. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.

CLAIMS:

1. A computer-implemented method of generating respective branch predictions, in a microprocessor, for first and second branch instructions, both of which are indexed by a first instruction pointer, the method comprising the steps of:

generating a first branch prediction for the first branch instruction; and

if the first branch is predicted as being not taken, then:

initiating a recycle stall during which the first instruction pointer is stalled;

propagating the first branch prediction to an instruction buffer;

generating a second branch prediction for the second branch instruction; and

propagating the second branch prediction to the instruction buffer.

10. The method of claim 1 wherein the microprocessor includes an instruction cache having an instruction cache line, indexed by the first instruction pointer, storing both the first and second branch instructions, the recycle stall being initiated if the both the first and second branch instructions are identified as being branch instructions.

13. Apparatus for generating respective branch predictions for first and second branch instructions, both indexed by a first instruction pointer, the apparatus comprising:

dynamic branch prediction circuitry to generate a branch prediction based on the outcome of previous branch resolution activity;

static branch prediction circuitry to generate a branch prediction based on static branch prediction information; and

prediction output circuitry, coupled to the both the dynamic and static branch prediction circuitry, to output the respective branch predictions for the first and second branch instructions in first and second clock cycles; and

recycle stall circuitry to initiate a recycle stall, during which the first instruction pointer is stalled, when the instruction pointer indexes the first and second branch instructions and when the first branch instruction is predicted as being not taken.

17. Apparatus for performing branch predictions for first and second branch instructions, both indexed by a first instruction pointer, the apparatus comprising:

branch prediction circuitry to generate first and second branch predictions for the first and second branch instructions respectively;

prediction output control circuitry, if the first branch instruction is predicted as being not taken, to:

the first branch prediction to an instruction buffer in a first clock cycle and in response to a recycle stall during which the first instruction pointer is stalled; and

propagate the second branch prediction to the instruction buffer in a second clock cycle.

18. The apparatus of claim 17 wherein the prediction output control circuitry includes recycle stall circuitry to initiate a recycle stall, during which the first instruction pointer is stalled, when the instruction pointer indexes the first and second branch instructions and when the first branch instruction is predicted as being not taken.

22. The apparatus of claim 18 including an instruction cache having an instruction cache line, indexed by the first instruction pointer and storing both the first and second branch instructions, the recycle stall circuitry initiating a recycle stall when the both the first and second branch instructions are identified as being branch instructions.

32. A computer-implemented method of generating respective branch predictions for first and second branch instructions, both indexed by a first instruction pointer, the method comprising the steps of:

determining whether the first branch instruction can be dynamically predicted based on the outcome of previous branch resolution activity;

if the first branch instruction cannot be dynamically predicted, determining whether the second branch instruction can be dynamically predicted based on the outcome of previous branch resolution activity; and

if the second branch instruction can be dynamically predicted, predicting the first branch instruction to be not taken.

33. A computer-implemented method of generating respective branch predictions, in a microprocessor, for first and second branch instructions, both of which are indexed by a first instruction pointer, the method comprising the steps of:

generating a first branch prediction for the first branch instruction; and

if the first branch is predicted as being not taken, then:

in a first clock cycle, initiating a recycle stall during which the first instruction pointer is stalled;

propagating the first branch prediction to an instruction buffer in response to the initiation of the recycle stall;

generating a second branch prediction for the second branch instruction;

terminating the recycle stall; and

in a second clock cycle, propagating the second branch prediction to the instruction buffer in response to the termination of the recycle stall.

L19: Entry 5 of 7 File: USPT Feb 4, 1997

DOCUMENT-IDENTIFIER: US 5600837 A
TITLE: Multitask processor architecture having a plurality of instruction pointers

Abstract Text (1):
A processor architecture for executing a current task among a plurality of possible tasks. The architecture includes: a plurality of instruction pointers respectively associated with the tasks and each storing the address of the current instruction to be executed of the associated task, only one of these pointers being enabled at a time to supply an address to the memory; a priority level decoder including circuitry for assigning a predetermined priority level to each request signal and for enabling the instruction pointer associated with the active request signal having the highest priority level; and a mechanism for incrementing the content of the enabled instruction pointer and for reinitializing it at the start address of the associated program when its content reaches the end address of the associated program.

Brief Summary Text (14):
These objects are achieved with a processor architecture for executing a current task from among a plurality of possible tasks, including: a memory for storing instructions of tasks; a processing unit coupled to the memory to receive therefrom successive instructions to be processed; a mechanism for providing active request signals associated respectively with the tasks that request to be served; a plurality of instruction pointers respectively associated with the tasks and each containing the address of the current instruction to be executed of the associated task, one of these pointers being enabled at a time to supply its content as an address to the memory; a priority level decoder including circuitry for assigning a predetermined priority level to each request signal and for enabling the instruction pointer associated with the active request signal having the highest priority level; and a mechanism for incrementing the content of the enabled instruction pointer and for reinitializing it at the start address of the associated program when its content reaches an end address of the associated program.

Brief Summary Text (15):
According to an embodiment of the invention, each instruction includes a command field that is provided to the processing unit and an instruction type field that is provided to a prefix decoder. The prefix decoder includes a mechanism for authorizing a new instruction pointer to be enabled by the priority level decoder, if the type field of the current instruction has a first predetermined value, and includes a mechanism for initializing the content of the enabled instruction pointer to the start address of the current task if the type field of the current instruction has a second predetermined value.

Brief Summary Text (16):
According to an embodiment of the invention, the prefix decoder includes a mechanism to inhibit the incrementation of the enabled instruction pointer if the type field has a third predetermined value, so that the current instruction is successively executed several times, the number of executions being determined by this third value.

Detailed Description Text (5):
In a conventional processor, the address at which the instruction is fetched is contained in a single instruction pointer. The content of this instruction pointer has to be saved and restored when the execution of a program is interrupted and is resumed subsequently.

Detailed Description Text (6):
In contrast, the present invention includes a plurality of instruction pointers IP1, IP2 . . . IPn each corresponding to a possible task to be executed. Each of these pointers is enabled by a respective enable line EN1-ENn so that the pointer may provide its content as an address of an instruction to be fetched for processing unit 16. The enable lines EN are controlled by a priority level decoder 18, which receives one request line REQ from each device 10 that may request processing by the processor. Each request line, and hence each device 10, is assigned a priority level. The same priority level may be assigned to several tasks. As such, decoder 18 must enable only one pointer IP at a time. This may be done, for example, by internally assigning different priority levels to these tasks and by circularly shifting these internal priority levels each time an instruction or a group of instructions is executed.

Detailed Description Text (8):
Programs are stored in the ROM or RAM at distinct locations and respectively correspond to tasks that may be executed. ROM may store basic programs that may be executed in any system of the architecture of FIG. 1. Similarly, RAM may store programs that are selected by a user and loaded from a floppy disk or hard disk, for example. The possibility for a user to load programs in RAM is particularly useful, if the system operates on a modular basis, in which the user may add or remove devices to be controlled on the system bus D, A, CTRL. The maximum number of tasks that the system can control is equal to the number n of instruction pointers IP provided in the processor.

Detailed Description Text (9):
The system operates as follows. Upon powering-on, each instruction pointer IPi (i=1, 2 . . . n) is initialized at a start value IPi.sub.0 corresponding to the start address of the associated program. These start addresses can, for example, be stored in a ROM table or on a disk and be transferred into the respective instruction pointers by a boot program, or by an operating system. Additionally, a priority level must be assigned to each program by assigning the rank of an instruction pointer to the associated program, for example. For this purpose, priority decoder 18 is either fixed or programmed at power-on. A programmable decoder comprises, for example, a multiplexer whose output, providing the instruction pointer enable signals, is selected among the contents of registers by the request signals.

Detailed Description Text (10):
Once the system is initialized, requests on lines REQ will likely appear. The priority decoder 18 will enable the instruction pointer IP associated with the request having the highest priority. The instruction processor 16 receives (or loads) the instruction located at the address indicated by the enabled instruction pointer and executes this instruction. Once the instruction is executed, an incrementation circuit 20 increments the enabled instruction pointer. This provides the address of the next instruction to be executed by the instruction processor 16. The incrementation circuit 20 acts only on the enabled pointer. That is, the contents of the other pointers are not modified. The representation (+1) of the incrementation circuit 20 is symbolic. The instructions to be executed are not necessarily located at successive addresses, for example when the instructions have different lengths or when jump or sub-program call instructions are executed. In such a case, the enabled instruction pointer is incremented or decremented by the adequate value, as is known in conventional processors. In most existing microprocessors, the value by which the instruction pointer is incremented or

decremented is included within the instructions.

Detailed Description Text (11):
The rank (i) of an instruction pointer IP is used to designate the associated program (or task) and request. A task may correspond to a plurality of looped executions of the associated program, or to the execution of a portion of the program. The task may also be continuous. Thus, the end of a program does not always correspond to the end of the associated task, and vice versa. The end of the task occurs when its request disappears. At such time, the task is said to be served.

Detailed Description Text (13):
Once the task i is served, request i is disabled, and decoder 18 enables the instruction pointer corresponding to a new maximum priority task. The new task may be a task that was suspended by task i, e.g., task i-1, may be an entirely new task. The associated program is executed immediately. If it is a suspended task, it starts from its suspension point. If it is a program corresponding to a task that has not been started yet, it starts at the starting address.

Detailed Description Text (15):
The operations of the instruction processor 16, of decoder 18, and of the incrementation circuit 20 are synchronized by a common clock signal CK provided by a clock circuit 23 which synchronizes the succession of instruction cycles. The incrementation of the pointers can be done either at the end, or at the beginning of each instruction cycle.

Detailed Description Text (20):
The prefix decoder 22 provides to the priority decoder 18 a signal NEXTEN that, if the prefix is at a specific value, inhibits the enabling of a new instruction pointer, even if a request having higher priority than that of the current program appears.

Detailed Description Text (21):
FIG. 2 represents an alternative embodiment of a multitask processor architecture according to the invention. Same elements as in FIG. 1 are designated with same reference numerals. This architecture is adapted to a multitask system having predefined tasks. The programs corresponding to the tasks are stored in a ROM 24 that is independent of the system bus D, A, CTRL. The content of the enabled instruction pointer IPi is directly provided to the address input of ROM 24 instead of being provided to the address bus A. The instruction processor 16 is coupled, as previously, to the system bus D, A, CTRL but is modified to directly receive instructions from ROM 24. Such a modification of a conventional processor may be easily achieved by those skilled in the art.

Detailed Description Text (24):
The prefix decoder 22 includes a down-counter of instruction cycles that is initialized by a number within prefix I1. This number can be a number selected among several constant numbers that are encoded by the prefix decoder or a number N provided by the data bus D. This number N is calculated and stored in the memory, or is provided by the served device 10. When such a loop instruction is executed, the prefix decoder 22 inhibits the incrementation circuit 20 during the desired number of instruction cycles. As a result, the loaded instruction is executed as many times as desired.

Detailed Description Text (26):
In the embodiment of FIG. 2, each instruction pointer IP includes two registers: one contains the address of the instruction to be executed of the associated program, and the other contains the start address IP.sub.0 of this program. Several solutions to provide values IP.sub.0 are available. For example, ROM 24 can include a table of start addresses, and these addresses may be written in the corresponding

registers during initialization. If the programs in the ROM 24 are identical from one system to another, their start addresses IP.sub.0 may be hard wired. This last solution avoids having to couple pointers IP to the system bus, which simplifies the processor structure.

Detailed Description Text (27):
If prefix I1 indicates that the instruction being executed is the last one, once the instruction is executed decoder 22 provides a signal INIT to the group of instruction pointers IP, which causes the enabled pointer only to be initialized with the program start address.

Detailed Description Text (37):
Such a transfer operation consists in a looped execution, as indicated above, of a RAM read instruction, the number of loops being determined by the prefix I1 of the read instruction. During each execution of the read instruction, RAM 14 provides on bus D a datum that is immediately transferred into the FIFO. The writing operations of data into the FIFO are synchronized with edges of the clock signal CK.

Detailed Description Text (39):
Such a transfer operation consists in a looped execution of a RAM write instruction, the number of loops being determined by the prefix I1 of the write instruction. Once the acknowledgement signal of the FIFO is activated, the FIFO provides its data on the bus at the rate of the clock signal CK, which is also the execution rate of the write instructions. Thus, each datum provided by the FIFO on the bus is immediately written in the RAM.

Detailed Description Text (40):
In order to determine the address in RAM 14 from which packets of data must be transferred, it is possible, for example, to update a pointer stored in RAM. The instruction processor 16 includes an address register AR containing the address at which a transfer (read or write) operation is carried out. The beginning of a transfer program of a packet of data includes an instruction that writes in this address register the content of the pointer. The subsequent instructions of the program are, for example, instructions to adequately modify the content of the address register at each transfer instruction. This adequate modification may consist in an incrementation or in a more complex calculation, for example a recursive operation.

CLAIMS:

1. A processor architecture for serving a current task from among a plurality of tasks, including:

a memory for storing instructions of a plurality of programs respectively corresponding to the plurality of tasks, the memory being connected to a data bus and an address bus;

an instruction processor coupled to the data bus to sequentially receive instructions from the memory to be processed at the rate of instruction cycles;

request lines of different priority levels associated respectively with the tasks, each request line being asserted as soon as the associated task is requesting to be served;

a plurality of instruction pointers respectively associated with the tasks, each containing an address of a current instruction of the associated task to be executed, and one of the instruction pointers providing its content on said address bus at each instruction cycle as long as an associated enable line is asserted, whereby a corresponding instruction is provided to the instruction processor and executed at each instruction cycle;

a priority level decoder connected to the request lines and to the enable lines of the instruction pointers, the priority encoder asserting the enable line of the instruction pointer corresponding to the asserted request line of highest priority level; and

an incrementation circuit connected to the instruction pointers to modify the instruction pointer associated with the asserted enable line at each instruction cycle to point to the next instruction of the associated task.

2. The architecture of claim 1, further comprising a prefix decoder and wherein each instruction includes a command field that is provided to the instruction processor and an instruction type field that is provided to the prefix decoder, the prefix decoder including:

means for authorizing a new instruction pointer to be enabled by the priority level decoder if the type field of the current instruction has a first predetermined value; and

means for initializing the content of the enabled instruction pointer to the start address of the current program if the type field of the current instruction has a second predetermined value.

9. A multitask processor for serving tasks associated with a plurality of devices, each device being coupled to a system bus, and each device providing a request signal when that device needs to be served, the processor comprising:

a first memory for storing a plurality of sets of instructions, each set of instructions being associated with one of the tasks;

a processing unit for executing the instructions at the rate of instruction cycles, the processing unit being coupled to the system bus;

a plurality of instruction pointers, each instruction pointer being associated with a task and holding an address of a current instruction in the first memory associated with the task, wherein when an address from an instruction pointer is provided to the first memory the current instruction associated with the task is provided to the processing unit;

means for prioritizing the request signals, including means for enabling an instruction pointer associated with the request signal having highest priority to provide its address to the first memory; and

means for updating at the rate of instruction cycles the instruction pointer that is enabled by the means for enabling so that the enabled instruction pointer is updated to point to the next instruction of the associated task.

10. The processor of claim 9 further including a prefix decoder, wherein each instruction includes a type field that is received by the prefix decoder, the prefix decoder having means for authorizing the means for prioritizing to enable a new instruction pointer, if the type field has a first predetermined value.

11. The processor of claim 10 wherein the prefix decoder includes means for inhibiting the means for updating from updating the instruction pointer.

12. The processor of claim 10 wherein the instruction pointers also store start addresses of the associated tasks and wherein the prefix decoder includes means for initializing the instruction pointer such that the current address is initialized to the start address of the associated task.

15. A method of serving a plurality of tasks associated with a plurality of devices, each device indicating that service is required by asserting a request signal on an associated request line, each of the request lines having an associated priority level, the method comprising the steps of:

a) recognizing a request line having an asserted request signal and being of the highest priority of the request lines having an asserted request signal;

b) enabling an instruction pointer from a plurality of instruction pointers, the enabled instruction pointer being associated with the recognized request line;

c) providing the contents of the enabled instruction pointer as an address to a program memory having instructions;

d) providing the instruction located at the address associated with the enabled instruction pointer to a processing unit;

e) the processing unit executing the instruction that is provided at the rate of instruction cycles; and

f) updating the enabled instruction pointer at the rate of instruction cycles so that the instruction pointer points to the next instruction of the associated task.

16. The method of claim 15 wherein the each instruction includes a type field and wherein the method further comprises a step (g) of determining whether the type field has a first predetermined value and wherein step (b) enables a new instruction pointer, if step (g) determines that the type field has the first predetermined value.

17. The method of claim 16 further comprising the step of (h) inhibiting the updating of the instruction pointer in step (f) if the type field has a second predetermined value.

(2) Detail of the Synchronization mode

First Hit Fwd Refs

End of Result Set

Generate Collection Print

L35: Entry 11 of 11 File: USPT Aug 22, 1989

DOCUMENT-IDENTIFIER: US 4860285 A
TITLE: Master/slave synchronizer

Abstract Text (1):
A synchronizer (100, 102, 104, 106) is disclosed operable in a variety of modes. In a Master/Slave mode the synchronizer receives synchronizing clock signals from a device to which it is a "slave" and generates therefrom synchronizing clock signals to a device to which it is a "master". In a Slave/Slave mode the synchronizer receives synchronizing clock signals from two devices to which it is a slave. In this mode the synchronizer can buffer misalignment between the clocks and report their phase difference for corrective action. In a Slave mode, the synchronizer only receives a synchronizing clock signal. A data-routing multiplexer (50, 108, 110) is employed in conjunction with the synchronizer which allows five devices to be connected to the synchronizer. Signals may be routed between any of the devices. Buffers (112, 120, 122) internal to the data-routing multiplexer perform the frame alignment function.

Brief Summary Text (13):
When the M/S DSC is operating in the Slave/Slave mode, the synchronizing clock of the PCM Highway and the synchronizing clock of the "S" interface can be asynchronous. Data streams synchronized by these two clocks can only be connected if their clock frequencies are the same or if occasional loss of data, or doubling of data, can be tolerated. Data lost or doubled because of asynchronous clocks is said to result from clock "slip". The master/slave synchronizer of the instant invention provides buffering of data between digital data streams synchronized by these asynchronous clocks, measurement of the phase between the clocks, and detection of clock slip which can cause corruption of data. A microprocessor, used in conjunction with the master/slave synchronizer, can digitally control the clock frequency of the PCM Highway.

Detailed Description Text (78):
(3) unsynchronized or synchronized depending on whether one or two clocks, respectively, are operating.

Detailed Description Text (80):
In normal operation the state machine 104 will start up in the IDLE state 200. When activity is detected on a clock, the state machine 104 passes to the appropriate start-up state, U1A for CLOCK1 only, or U2B for CLOCK2 only. State machine 104 then alternates between the A and B modes until activity is detected on the other clock. State machine 104 then passes to an appropriate synchronized mode (S1A, S2A, S1B, or S2B). If an access occurs too later for proper data transfer, the machine sets the slip interrupt signal.

Detailed Description Text (84):
U1A state 202: Only CLOCK1 is active, and less than 1/2 frame has passed since the last CLOCK1 strobe. If CLOCK2 becomes active, the state machine will synchronize in the A mode. PHASE 126 and CLOCK2 Transmit Registers are zero in this state.

h e b b g e e e f c e e ge

Detailed Description Text (85):
U1B state 204: Only CLOCK1 is active, and more than 1/2 frame has passed since the last CLOCK1 strobe. If CLOCK2 becomes active, the state machine will synchronize in the B mode. PHASE and CLOCK2 Transmit Registes are zero in this state.

Detailed Description Text (86):
U2A state 206: Only CLOCK2 is active, and more than 1/2 frame has passed since the last clock 2 strobe. If CLOCK1 becomes active, the state machine will synchronize in the A mode. PHASE and CLOCK1 Transmit Registers are zero in this state.

Detailed Description Text (87):
U2B state 208: Only CLOCK2 is active, and less than 1/2 frame has passed since the last CLOCK2 strobe. If the CLOCK1 becomes active, the state machine will synchronize in the B mode. PHASE and CLOCK1 Transmit Registers are zero in this state.

Detailed Description Text (88):
S1A state 210: The state machine 104 is synchronized in the A mode and the last access was on CLOCK1.

Detailed Description Text (89):
S2A state 212: The state machine 104 is synchronized in the A mode and the last access was on CLOCK2.

Detailed Description Text (90):
S1B state 214: The state machine 104 is synchronized in the B mode and the last access was on the CLOCK1.

Detailed Description Text (91):
S2B state 216: The state machine 104 is synchronized in the B mode and the last access was on CLOCK2.

Detailed Description Text (98):
Upon receipt of a clock flag in the IDLE state, the machine moves to an unsynchronized state (U1A or U2B) and clears the timer (T.rarw.O). The UXX states are used when only one clock is active. The number after the "U" indicates which clock is active; e.g., U1X indicates that only CLOCK1 is active (X=A or B).

Detailed Description Text (103):
When the second clock occurs, the machine moves to the appropriate synchronized state (SXX). The number after the "S" tells which clock occurred last. The letter at the end indicates A mode or B mode. For example, S1A means synchronized in A mode with CLOCK1 as the last event. In moving from an unsynchronized to a synchronized state, the mode (A or B) remains unchanged.

Detailed Description Text (124):
With reference now to FIG. 8, several examples of the synchronizing function performed by the master/slave multiplexer of the present invention are illustrated in data-transfer timing diagrams. Each example shows the use of either CLOCK1 or CLOCK2 as the synchronizing clock for accesses to a "CHANNEL1" and a "CHANNEL2"; either of which can be any of the four channels described in Table III.

Detailed Description Text (126):
In accordance with FIG. 8, state-machine 104 enters U1A "unsynchronized" state 202, TIMER 108 is cleared (T.rarw.0). PHASE 126 is set to ZERO and Transmit Register2 122 is cleared. At the generation of the TIMER MIDPOINT (TM) signal a TRAN1-2 signal is generated by state machine 104 causing movement of data to MUX CHANNEL 2 as indicated by vertical line 226 connected to CHANNEL 2 access-line 222 and state-machine 104 enters U1B unsynchronized state 204.

Detailed Description Text (129):
The occurrence of CLOCK2 sets CF2 and synchronized S2A state 212 is entered in accordance with FIG. 8. The "2" indicating that CLOCK2 occured most-recently; the "A" indicating positive phase. In transiting from unsynchronized state U1A to synchronized state S2A the mode A remains unchanged. Normally, the mode changes infrequently and the state machine 104 either alternates between S1A and S2A has shown on access lines 220 and 222 to vertical lines 234, 236, 238, 240, 242 and 244 or between S1B and S2B, as each clock occurs.

Detailed Description Text (131):
With reference now to the second example illustrated in FIG. 8, on channel-access lines 246 and 248, the state machine 104 is started in Idle state 200. A CLOCK2 signal is requested to access CHANNEL2, as indicated by vertical line 250. The CF2 flag is thus set and state-machine 104 enters U2B unsynchronized state 208. A TM signal is generated by TIMER 108 and U1A state 206 is entered. A TRANS1-2 signal is generated by state-machine 104, in accordance with FIG. 7 and a transfer is used for CHANNEL2 to CHANNEL1 shows by vertical line 252.

CLAIMS:

7. A method of synchronizing transmission of data through a multiplexer, having a plurality of ports, responsive in a first mode to first and second synchronizing clock signals and in a second mode to an active clock signal being said first or said second synchronizing clock signals, comprising the steps:

(a) synchronously transferring said data between a first buffer and a first port using said first synchronizing clock signal when in said first mode, and using said active synchronizing clock signal, when in said second mode;

(b) routing said data between said first buffer and a second buffes; and

(c) synchronously transferring said data between said second buffer and a second port using said second synchronizing clock signal, when in said first mode, and using said active synchronizing clock signal when in said second mode.